INITIAL PROCESSING LOAD SCATTERING SYSTEM FOR MULTI-CPU CONSTITUTION SYSTEM

Patent number: JP10011412
Publication date: 1998-01-16
Inventor: AMAGASAKI KENJI
Applicant: NIPPON DENKI IDO TSUSHIN KK
Classification:
- international: G06F15/177; G06F15/16
- european:
Application number: JP19960181663 19960624
Priority number(s): JP19960181663 19960624

View INPADOC patent family

Abstract of **JP10011412**

PROBLEM TO BE SOLVED: To quickly start the scattering system without impairing the starting order by dividing the initial processing into blocks of every function that can be processed in parallel to each other and assigning these processing of divided blocks to all CPUs. SOLUTION: The initial processing is divided into blocks of every function that can be processed in parallel to each other. For instance, the check/clear processing of local memories 2-1 to 2-5 used for initial processing in common to all CPUs and the processing which loads the programs in the memories 2-1 to 2-5 from the system data are set in a block A. Then the processing is set in a block B to transfer the information contents of a common memory 3-2 under waiting to a backup memory 6. When the system is reset, the CPU 2-1 to 2-5 are also reset. Then the block A of the processing common to all CPUs is carried out. When the processing of the block A is completed, a CP# setting processing is carried out based on a CP# management table that is set in the memory 6.

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－11412

(43)公開日　平成10年(1998)１月16日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	15/177			Ｇ０６Ｆ	15/16	４２０Ｓ
	15/16	３８０				３８０Ｚ

審査請求　有　　請求項の数３　ＦＤ　（全　９　頁）

(21)出願番号　特願平8－181663

(22)出願日　平成８年(1996)６月24日

(71)出願人　390000974
日本電気移動通信株式会社
横浜市港北区新横浜三丁目16番８号　（ＮＥＣ移動通信ビル）

(72)発明者　尼崎　健治
神奈川県横浜市港北区新横浜三丁目16番８号　日本電気移動通信株式会社内

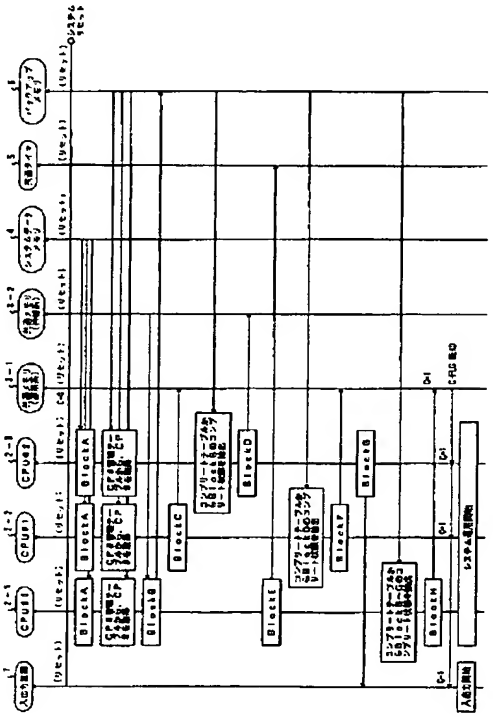
(74)代理人　弁理士　高橋　友二

(54)【発明の名称】　マルチＣＰＵ構成システムにおける初期処理負荷分散方式

(57)【要約】

【課題】　システムの立ち上げ順序をそこなうことなく、迅速なシステムの立ち上げを可能とするマルチＣＰＵ構成システムにおける初期処理負荷分散方式を提供する。

【解決手段】　マルチＣＰＵにて構成されるシステムの起動時に行われる初期処理負荷を分散する方式であって、初期処理をパラレルに処理できる機能ごとにいくつかのブロックに分割し、実装されているすべてのＣＰＵ２－１～２－３にブロック化した処理を割り当てることにより、初期処理の負荷を分散する。



【特許請求の範囲】
【請求項１】　マルチＣＰＵにて構成されるシステムの起動時に行われる初期処理負荷を分散する方式であって、初期処理をパラレルに処理できる機能ごとにいくつかのブロックに分割し、実装されているすべてのＣＰＵにブロック化した処理を割り当てることにより、初期処理の負荷を分散することを特徴とするマルチＣＰＵ構成システムにおける初期処理負荷分散方式。
【請求項２】　前記ＣＰＵに対するブロック化した処理を割り当てるに際し、コンプリート状態を確認することで、システムの立ち上げの順序をそこなわないようにした請求項第１項記載のマルチＣＰＵ構成システムにおける初期処理負荷分散方式。
【請求項３】　前記ブロック化した初期処理が、各ＣＰＵ共通初期処理のローカルメモリのチェック及びクリア処理、システムデータからプログラムをローカルメモリへロードする処理、共通メモリ（待機系）の情報内容をバックアップメモリへ転送する処理、共通メモリ（運用系）のチェック及びクリア処理、共通メモリ（待機系）のチェック及びクリア処理、共通タイマのタイマクリア処理、共通データテーブルの初期化処理、入出力装置の初期化処理、初期処理完了フラグのＯＮ処理のいずれかである請求項第１項または第２項記載のマルチＣＰＵ構成システムにおける初期処理負荷分散方式。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、マルチＣＰＵ構成システムにおける起動時間の短縮化を図るための初期処理負荷分散方式に関するものである。
【０００２】
【従来の技術】従来から、処理時間の短縮を図るために、処理を分割したグループＣＰＵで並列に行なうマルチＣＰＵ構成システムは、例えば特開平５－１２０２４５号公報に開示されているように良く知られている。図１はそのような従来のマルチＣＰＵシステムの構成の一例を示す図である。なお、本発明の対象となるマルチＣＰＵ構成システムは、図１に示す従来のものと同様である。
【０００３】図１に示す例において、マルチＣＰＵ構成システム１は、所望の実装枚数、ここでは５枚のＣＰＵ２－１～２－５と、動的に変化するデータを格納しておく運用系および待機系の２系列からなる共通メモリ３－１、３－２と、プログラム，システムデータ等の静的なデータを格納しておくシステムデータメモリ４と、シーケンス管理上必要なタイマを管理する共通タイマ５と、共通メモリ３－１または３－２のデータをバックアップしておくためのバックアップメモリ６と、外部とのインターフェースを担う入出力装置７とから構成されている。なお、８－１～８－２は各ＣＰＵ２－１～２－５に設けられたローカルメモリである。
【０００４】図２は図１に示すマルチＣＰＵ構成システム１の起動時に行われる初期処理の状態を説明するための図である。図２において、まずシステムリセットが行われると、システムを構成する各装置がリセットされる。各々のＣＰＵ２－１～２－５は、各ＣＰＵが具備するローカルメモリ８－１～８－５のチェック・クリアを行い、システムデータメモリ４にあるプログラムのロードを行う。次に、運用系の共通メモリ３－１上にあるinitial-FLAG（初期処理フラグ、Ｉ－ＦＬＧと記す）の値を読み出し、このＩ－ＦＬＧの値が「０」であったならば、そのＣＰＵ、ここではＣＰＵ２－１がシステムの初期処理を担当するマスタＣＰＵとなる。また、このＩ－ＦＬＧの値が「１」であったならば、その他のＣＰＵ（ここではＣＰＵ２－５を例示する）は、マスタＣＰＵがシステムの初期処理を完了するまで待つスレーブＣＰＵとなる。
【０００５】マスタＣＰＵとなったＣＰＵ２－１は、順に、待機系の共通メモリ３－２のデータをバックアップメモリ６へ待避させる待避処理を行い、運用系共通メモリ３－１と待機系共通メモリ３－２のチェック・クリア処理を行ない、共通タイマ５のクリア処理を行ない、共通メモリ３－１上のデータテーブルの初期化を行ない、入出力装置７の初期化を行ない、共通メモリ３－１上にあるcomplete-FLAG（初期処理完了フラグ、Ｃ－ＦＬＧと記す）に「１」を設定する。そして、Ｃ－ＦＬＧが「１」になったことをスレーブＣＰＵ２－５と入出力装置７が認識後、システムが運用開始となる。
【０００６】
【発明が解決しようとする課題】上述した従来のマルチＣＰＵ構成システム１では、初期処理を１枚のＣＰＵ（マスタＣＰＵ）２－１のみで行っているため、プログラムのバージョンアップ、システムデータの変更等に伴うシステムリセットが行われた際、システムが再度運用を開始するまでの時間がかなりかかるという問題があった。また、システムが運転再開するまでの時間、ユーザに対してサービスを提供することができないため、顧客満足度が低下する問題もある。さらに、今後の良質なサービスの提供のためにプログラムおよび共通メモリの増大が見込まれており、一層初期処理時間が増大する可能性が高く、そのような場合上述した問題がより一層顕著になる。
【０００７】本発明の目的は上述した課題を解消して、システムの立ち上げ順序をそこなうことなく、迅速なシステムの立ち上げを可能とするマルチＣＰＵ構成システムにおける初期処理負荷分散方式を提供しようとするものである。
【０００８】
【課題を解決するための手段】本発明のマルチＣＰＵ構成システムにおける初期処理負荷分散方式は、マルチＣＰＵにて構成されるシステムの起動時に行われる初期処

理負荷を分散する方式であって、初期処理をパラレルに処理できる機能ごとにいくつかのブロックに分割し、実装されているすべてのＣＰＵにブロック化した処理を割り当てることにより、初期処理の負荷を分散することを特徴とするものである。

【０００９】具体的に本発明のマルチＣＰＵ構成システムにおける初期処理負荷分散方式は、マルチＣＰＵにて構成されるシステムの起動時に行われる初期処理負荷を分散する方式であって、初期処理をパラレルに処理できる機能ごとにいくつかのブロックに分割し、実装されているすべてのＣＰＵにブロック化した処理を割り当てることにより、初期処理の負荷を分散することを特徴とする。

【００１０】また、前記ＣＰＵに対するブロック化した処理を割り当てるに際し、コンプリート状態を確認することで、システムの立ち上げの順序をそこなわないようにしたことを特徴とする。

【００１１】さらに、前記ブロック化した初期処理が、各ＣＰＵ共通初期処理のローカルメモリのチェック及びクリア処理、システムデータからプログラムをローカルメモリへロードする処理、共通メモリ（待機系）の情報内容をバックアップメモリへ転送する処理、共通メモリ（運用系）のチェック及びクリア処理、共通メモリ（待機系）のチェック及びクリア処理、共通タイマのタイマクリア処理、共通データテーブルの初期化処理、入出力装置の初期化処理、初期処理完了フラグのＯＮ処理のいずれかであることを特徴とする。

【００１２】

【発明の実施の形態】以下、本発明の一実施例を図面を参照して説明する。図３は本発明のマルチＣＰＵ構成システムにおける初期処理負荷分散方式における初期処理のブロック化の状態を示す図である。図３では、図２に示す従来のマルチＣＰＵ構成システムの初期処理を分割して、処理できる機能ごとにブロック化を図っている。図３に示す例において、ブロックＡには、各ＣＰＵ共通初期処理のローカルメモリ８－１～８－５のチェック及びクリア処理およびシステムデータからプログラムをローカルメモリ８－１～８－５へロードする処理を設定し、ブロックＢには、待機系の共通メモリ３－２の情報内容をバックアップメモリ６へ転送する処理を設定し、ブロックＣには、運用系の共通メモリ３－１のチェック及びクリア処理を設定し、ブロックＤには、待機系の共通メモリ３－２のチェック及びクリア処理を設定し、ブロックＥには、共通タイマ５のタイマクリア処理を設定し、ブロックＦには、運用系の共通メモリ３－１内の共通データテーブルの初期化処理を設定し、ブロックＧには、入出力装置７の初期化処理を設定し、ブロックＨには、初期処理完了フラグＯＮを設定する。

【００１３】図４は本発明の初期処理分散方式を用いたマルチＣＰＵ構成システムの起動時に行われる初期処理の状態の一例を示す図である。図４に示す例では、ＣＰＵ２－１、２－２、２－３の３枚を実装した例を示している。まず、図４に示すマルチＣＰＵ構成システムにおいて、システムリセットが実行されると、ＣＰＵ２－１～２－３に対しリセットがかかる。その後、図３に示す各ＣＰＵ共通処理のブロックＡを実行する。ブロックＡの処理が終了した後、バックアップメモリ６上に設定されている図７に示すＣＰ＃管理テーブルを利用して、図８に示すＣＰ＃設定処理を実行する。

【００１４】図８に従ってＣＰ＃設定処理を説明すると、まず図７に示すＣＰ＃管理テーブルからＬＯＣＫを取得する（８０１）。ＬＯＣＫの内容が「ＯＦＦ」であれば、次の処理を行ない、「ＯＮ」であれば再度ＬＯＣＫの取得を行なう（８０２）。次に、ＬＯＣＫに「ＯＮ」を設定する（８０３）。そして、図７に示すＣＰ＃管理テーブルからＣＰ＃を取得し（８０４）、ＣＰ＃が５未満であるかの判断を行ない（８０５）、５未満であれば取得したＣＰ＃を１インクリメントし、テーブルに設定する（８０６）。５以上であれば次の処理を行なう。再度にＬＯＣＫを「ＯＦＦ」に設定する（８０７）。ここでは、ＣＰＵ２－１がＣＰ＃０に、ＣＰＵ２－２がＣＰ＃１に、ＣＰＵ２－３がＣＰ＃２に設定されている。

【００１５】次に、各ＣＰＵは取得したＣＰ＃に基づき、図５に示すブロック処理ディスパッチ（割当）一覧表によって、処理すべきブロックを判定する。すなわち、バックアップメモリ６上に設定されている図６に示すコンプリートテーブルを利用して、図９に示すコンプリート状態の設定処理を実行する。

【００１６】図９に従ってコンプリート状態設定処理を説明すると、ＣＰ＃０のＣＰＵ２－１は、図５に示す一覧表におけるＣＰＵ実装数３の欄のＣＰ＃０の部分を基にして、ブロックＢの図６に示すコンプリートテーブルのＬＯＣＫを取得する（９０１）。ＬＯＣＫの内容が「ＯＦＦ」であれば次の処理を行ない、「ＯＮ」であれば再度ＬＯＣＫの取得を行なう（９０２）。ＬＯＣＫに「ＯＮ」を設定する（９０３）。そして、ブロックＢの図６に示すコンプリートテーブルのコンプリート状態を取得し（９０４）、コンプリート状態が「未実施」であれば（９０５）、コンプリート状態に「実施中」を設定する（９０６）。コンプリート状態が「未実施」以外の場合は、異常処理を行なう。次に、ＣＰ＃に現在のＣＰ＃ここでは「０」を設定する（９０９）。最後にＬＯＣＫに「ＯＦＦ」を設定する（９１０）。その後、ＣＰ＃０のＣＰＵ２－１においてブロックＢの処理を行ない、処理終了後、図９のコンプリート状態設定処理に従って、コンプリート状態「実施中」（９０７）をコンプリート状態「完了」に設定する（９０８）。

【００１７】上記と同様に、ＣＰ＃１のＣＰＵ２－２はブロックＣの処理を行なう。ＣＰ＃２のＣＰＵ２－３

は、図５に示すブロック処理ディスパッチ一覧表に従ってブロックＤの処理を行なう様に判断できるが、コンプリート条件があるのでブロックＢのコンプリート状態が「完了」になった後に処理を開始する。

【００１８】ここで各ＣＰＵは、ブロックＡの後の一回目の処理が終了したので、再度、図７に示すＣＰ＃管理テーブルからＣＰ＃を取得し、図５に示すブロック処理ディスパッチ一覧表のＣＰＵ実装数３の部分を判定し、ＣＰ＃０のＣＰＵ２－１はブロックＥの処理を実行し、ＣＰ＃１のＣＰＵ２－２はブロックＤのコンプリート状態が「完了」になった後ブロックＦの処理を実行し、ＣＰ＃２のＣＰＵ２－３はブロックＧの処理を実行する。

【００１９】最後に、ＣＰ＃０のＣＰＵ２－１がブロックＢ～Ｇのコンプリート状態を判定し、すべて「完了」であれば、ブロックＨの処理を実行する。図５に示す各ＣＰ＃の処理終了後のＣＰＵ２－１～２－３とブロックＧ終了後の入力装置７は、共通メモ３－１上にあるＣ－ＦＬＧが「１」になるまで取得を行って、「１」になったことを認識後、システムが運用開始となる。

【００２０】

【発明の効果】以上の説明から明らかなように、本発明のマルチＣＰＵ構成システムにおける初期処理負荷分散方式によれば、初期処理がブロック化され複数のＣＰＵによってパラレルに処理実行されることにより、システムの立ち上げ順序をそこなうことなく、迅速なシステムの立ち上げが可能となる。

【図面の簡単な説明】

【図１】本発明の対象となるマルチＣＰＵ構成システムの一例の構成を示すブロック図である。

【図２】マルチＣＰＵ構成システムにおける従来の初期処理の状態を示す図である。

【図３】マルチＣＰＵ構成システムにおける初期処理のブロック化の一例を示す図である。

【図４】本発明の初期処理負荷分散方式を用いたマルチＣＰＵ構成システムにおける初期処理の状態を示す図である。

【図５】ブロック処理のディスパッチ一覧表の一例を示す図である。

【図６】コンプリートテーブルの一例を示す図である。

【図７】ＣＰ＃管理テーブルの一例を示す図である。

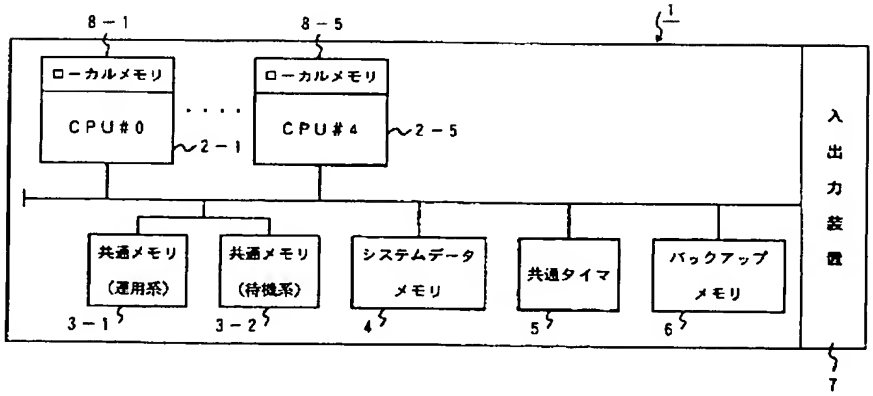
【図８】ＣＰ＃設定処理の一例を示すフローチャートである。

【図９】コンプリート状態設定処理の一例を示すフローチャートである。

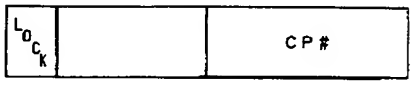
【符号の説明】

１　マルチＣＰＵ構成システム
２－１～２－５　ＣＰＵ
３－１，３－２　共通メモリ
４　システムデータメモリ
５　共通タイマ
６　バックアップメモリ
７　入出力装置

【図１】



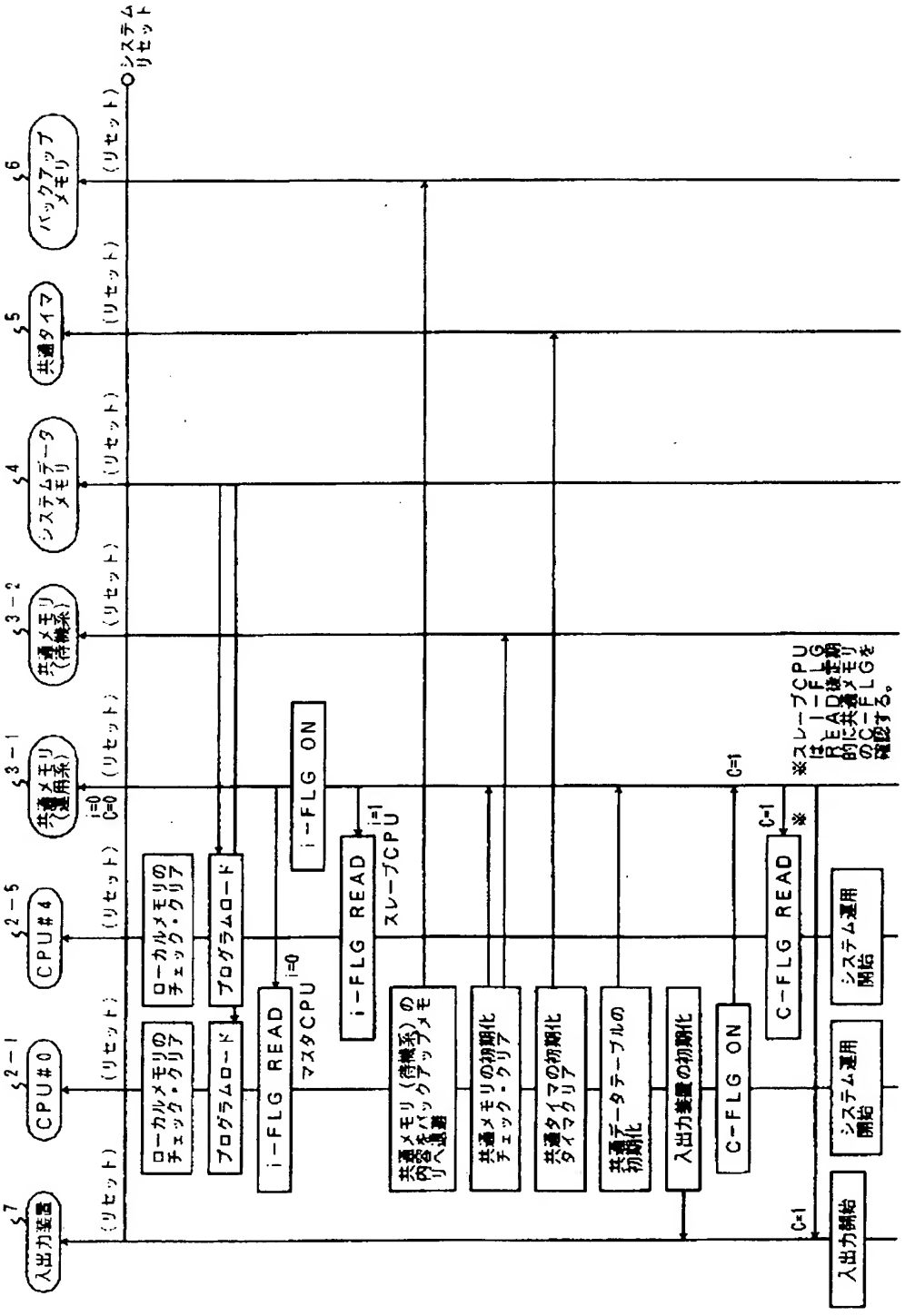
【図７】



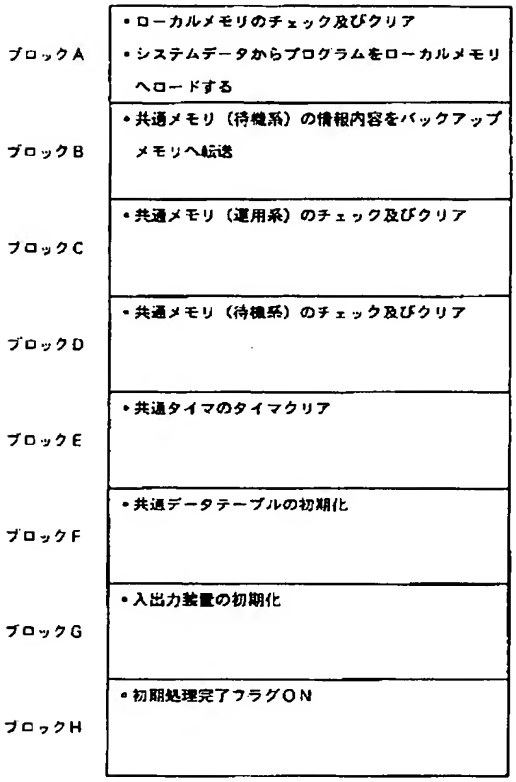
CP#：0～5

LOCK：OFF／ON（0／1）

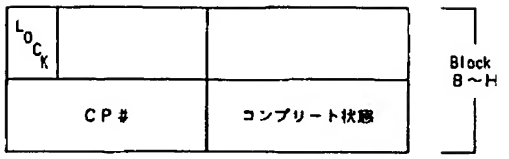
【図2】


システムリセット
(リセット)
バックアップメモリ
共通タイマ
システムデータメモリ
共通メモリ(待機系)
共通メモリ(運用系)
CPU#4
CPU#0
入出力装置
ローカルメモリのチェック・クリア
プログラムロード
i-FLG READ
マスタCPU
i-FLG ON
i-FLG READ
スレーブCPU
共通メモリ(待機系)の内容をイメージアップメモリへ退避
共通メモリの初期化チェック・クリア
共通タイマのタイマクリア
共通データテーブルの初期化
入出力装置の初期化
C-FLG ON
C-FLG READ
システム運用開始
システム運用開始
入出力開始
C=1
i=0
C=0
i=1
※スレーブCPUはC-FLG READ的に共通メモリのC-FLGを確認する。

【図３】



【図６】



CP#：0～4

LOCK：OFF／ON（0／1）

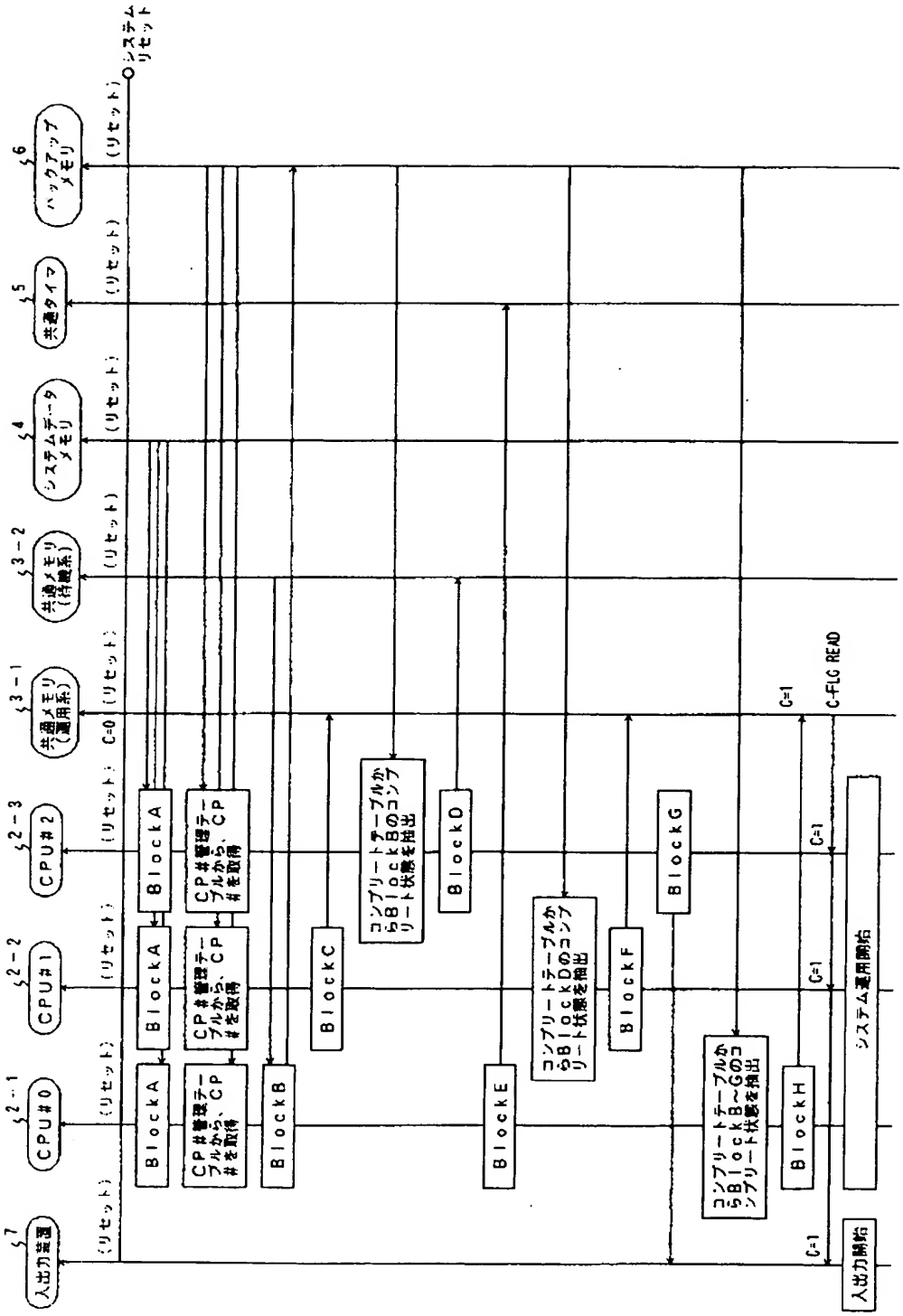
コンプリート状態：未実施／実施中／完了（00／0F／FF）

テーブル初期値：（LOCK＝OFF，コンプリート状態＝未実施，CP#＝0）

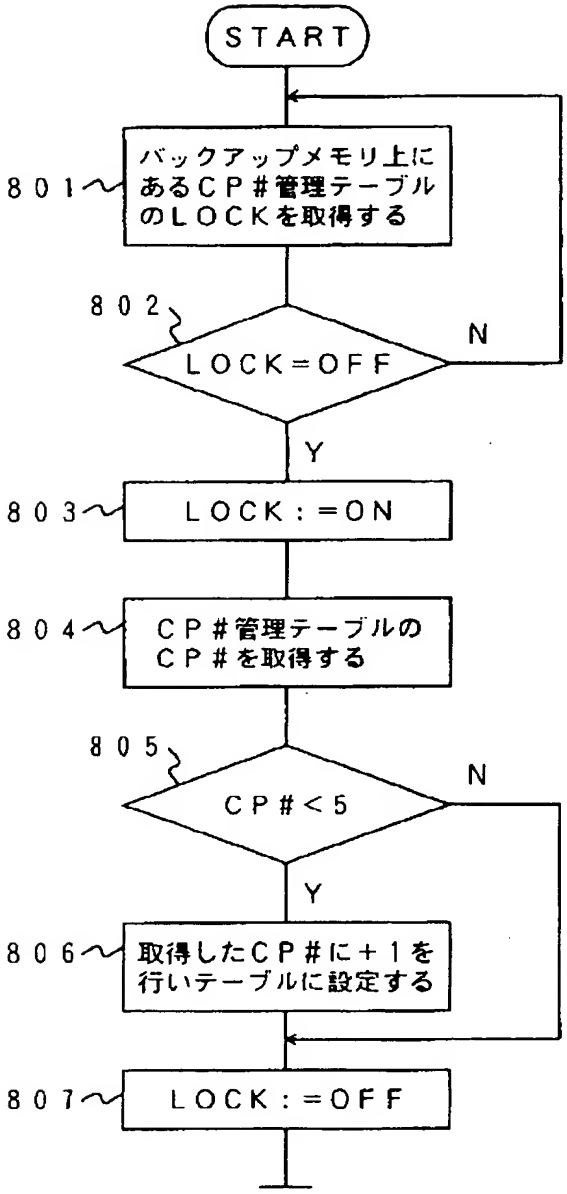
【図５】

CP# ＼ Block#	B	C	D	E	F	G	H	CPU実装数
CP0	○	○	○	○	○	○	○	1
CP0	○		○		○		○	2
CP1		○		○		○		
CP0	○			○			○	3
CP1		○			○			
CP2			○			○		
CP0	○				○			4
CP1		○				○		
CP2			○				○	
CP3				○				
CP0	○					○		5
CP1		○					○	
CP2			○					
CP3				○				
CP4					○			
コンプリート条件	なし	なし	BlockB コンプリート	なし	BlockD コンプリート	なし	BlockB～G コンプリート	

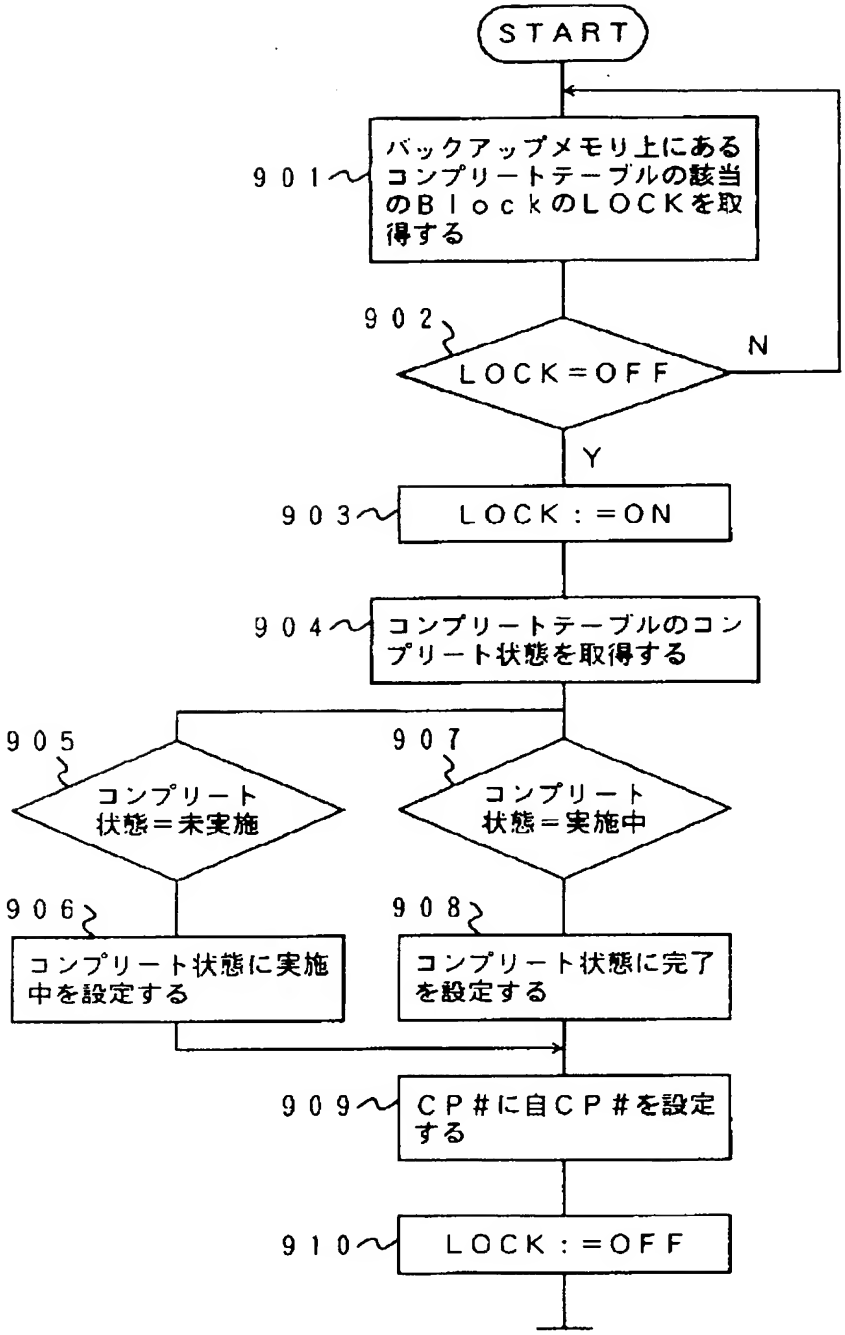
【図4】


システムリセット
(リセット)
ハックアップメモリ
§6
共通タイマ
§5
システムデータメモリ
§4
共通メモリ(待機系)
§3－2
共通メモリ(運用系)
§3－1
CPU#2
§2－3
CPU#1
§2－2
CPU#0
§2－1
入出力装置
§7
C=0
C=1
C-FLG READ
BlockA
BlockB
BlockC
BlockD
BlockE
BlockF
BlockG
BlockH
CP#管理テーブルから、CP#を取得
コンプリートテーブルからBlockBのコンプリート状態を抽出
コンプリートテーブルからBlockDのコンプリート状態を抽出
コンプリートテーブルからBlock B～Gのコンプリート状態を抽出
システム運用開始
入出力開始

【図8】


START
801
バックアップメモリ上にあるCP#管理テーブルのLOCKを取得する
802
LOCK=OFF
N
Y
803
LOCK:=ON
804
CP#管理テーブルのCP#を取得する
805
CP#<5
N
Y
806
取得したCP#に+1を行いテーブルに設定する
807
LOCK:=OFF

【図9】


START
901
バックアップメモリ上にあるコンプリートテーブルの該当のＢｌｏｃｋのＬＯＣＫを取得する
902
ＬＯＣＫ＝ＯＦＦ
N
Y
903
ＬＯＣＫ：＝ＯＮ
904
コンプリートテーブルのコンプリート状態を取得する
905
コンプリート状態＝未実施
907
コンプリート状態＝実施中
906
コンプリート状態に実施中を設定する
908
コンプリート状態に完了を設定する
909
ＣＰ＃に自ＣＰ＃を設定する
910
ＬＯＣＫ：＝ＯＦＦ